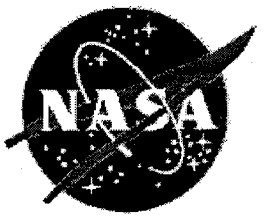
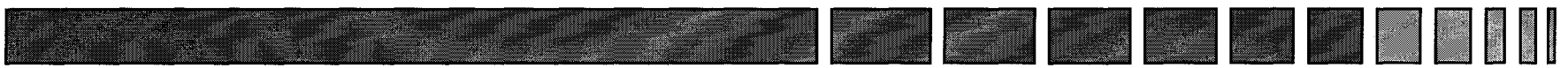
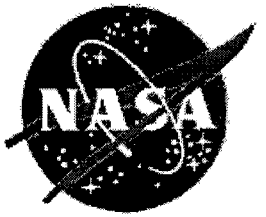




Formal Specification and Analytical Verification

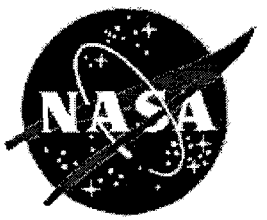
Presented by:

John D. Powell



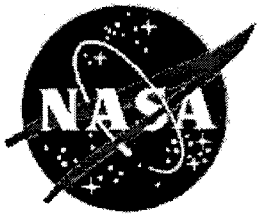
Contents

What is Formal Methods?



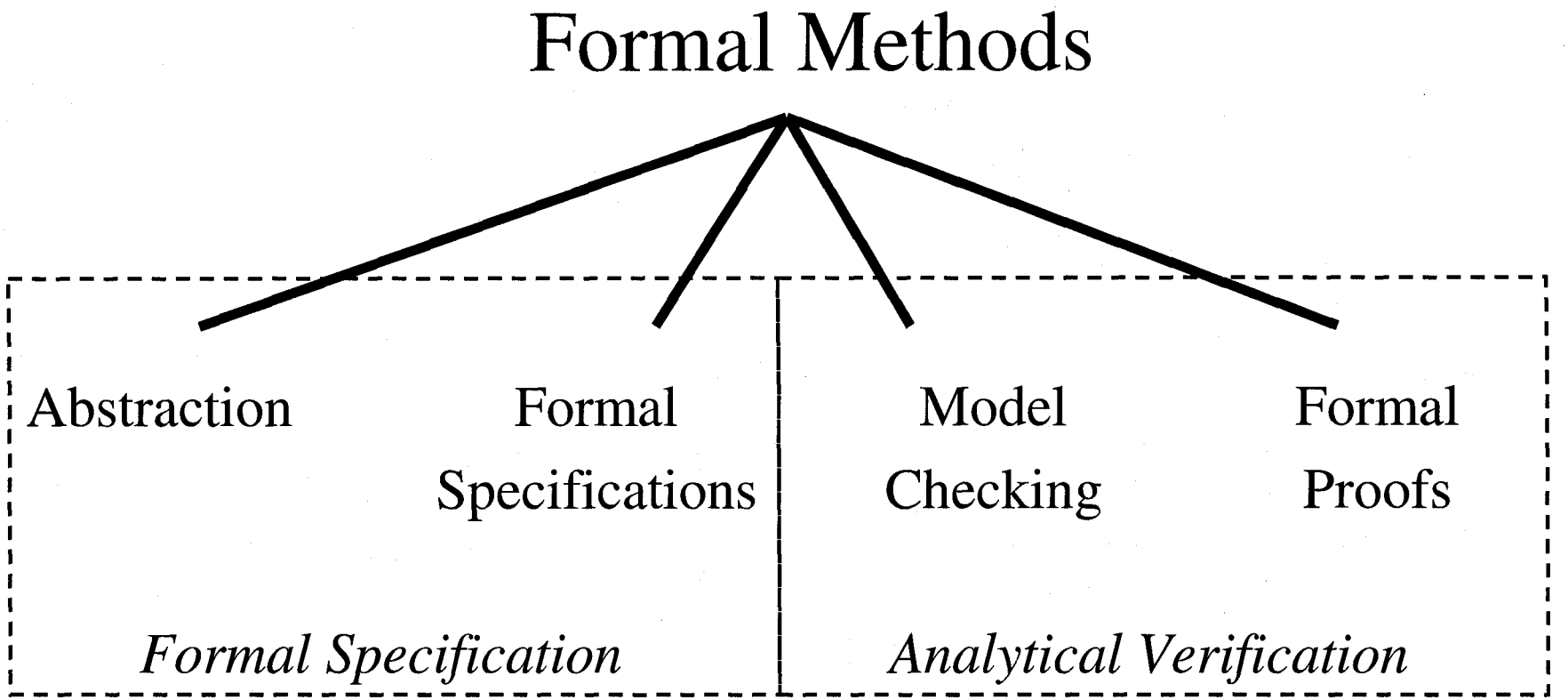
- Tools and Techniques based on formal logic and mathematics to specify and verify systems, software, and hardware
- Specification
  - » A precise "abstract" mathematical model of a component's specification
    - Symbolic Manipulation vs. Arithmetic Execution
  - » Formal methods help debug requirements & design specifications
- Verification
  - » Complement empirical methods such as traditional testing
  - » Fidelity between code and formal models can be checked via test scenarios
    - **Model checking component can generate counterexamples and help form equivalence classes for testing**

What is Formal Methods? (continued)

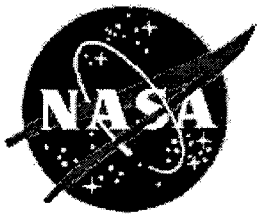


- The application of formal methods can include the following techniques:
  - » **Formal Specifications** allow the rigorous capture of the abstract model
    - **Abstraction** separates the central characteristics from irrelevant details
  - » **Analytical Verification** can prove properties exhaustively in models
    - Model Checking and/or Animation
      - Exhaustive state exploration of abstract model
      - Inductive analysis
    - Proofs
      - Deductive analysis

Formal Methods Concepts

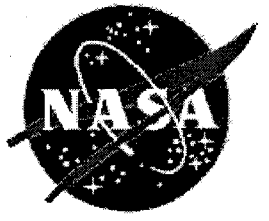


Abstraction



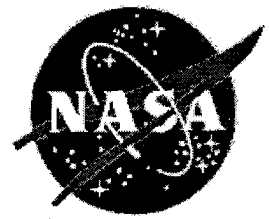
- Simplify and ignore irrelevant details
- Focus on and generalize important central properties and characteristics
- Avoid premature commitment to design and implementation choices

Formal Specifications



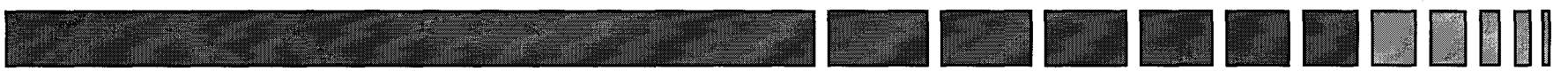
- Translation of a non-mathematical description (diagrams, tables, English text) into a formal specification language
- Concise description of high-level behavior and properties of a system
- Well-defined language semantics support formal deduction about specification

Model Checking



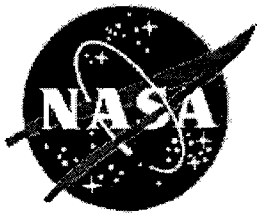
- Operational rather than analytic
- State machine model of a system is expressed in a suitable language
- Model checker determines if the given finite state machine model satisfies requirements expressed as formulas in a given logic
- Basic method is to explore all reachable paths in a computational tree derived from the state machine model

Formal Proofs



- Complete and convincing argument for validity of some property of the system description
- Constructed as a series of steps, each of which is justified from a small set of rules
- Eliminates ambiguity and subjectivity inherent when drawing informal conclusions
- May be manual but usually constructed with automated assistance

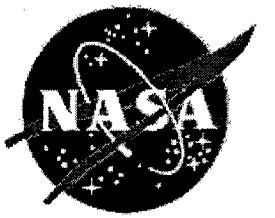
Written Requirement



Informal requirements expressed in English:

- A tank of cooling water shall be refilled when its low level sensor comes on. Refilling consists of adding 9 units of water to the tank.
- Notes:
  - The maximum capacity of the tank is 10 units of water.
  - From one reading of the water level to the next reading of the water level, 1 unit of water will be used.
  - The low level sensor comes on when the tank contains 1 unit of water or less.

Assigning Types



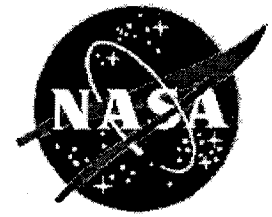
- The above statement contains several descriptions, including two key notions: the water level in the tank and the water usage. Formally, these notions can be modeled as follows (statements 1 and 2):

  1 ***level*** *is represented by a restricted integer type: a number between 0 and 10, inclusive*

  2 ***usage*** *is represented as the integer constant 1*

- That is, **level** describes an amount of water that the tank may hold at any point in time and **usage** describes the amount of water used during one cycle.

Function Description

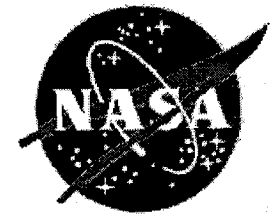


- The primary requirement is that 9 units of water will be added to the tank whenever the level is less than or equal to 1. This can be more precisely stated as (statement 3):

  *3 Function **fill** takes, as input, a water level and returns, as output, a water level. Given an input of L units of water, **fill** returns L+9, if L is one or less, otherwise it returns L.*

- That is, we claim that **fill(L)** accounts for any filling of water in the tank.

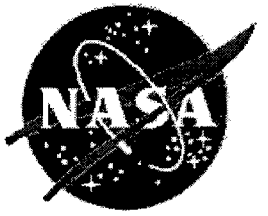
Properties



- A common sense property of this system is that, at the next cycle, the new water level will be the current water level, plus any amount that was added, minus the amount that was used. That is, given L as the current level of water, the level at the next cycle should be given by statement 4:

4 $level = L + fill(L) - usage$

Properties (continued)



- One approach to checking this specification is to ensure that each reference to a level of water is consistent with the definition of level, i.e., it should always be a number between 0 and 10. It turns out that the specification for fill given in 3 above is consistent with the definition of level *if* the following two logical statements are true:

5 *FORALL levels L*

(L <= 1) IMPLIES THAT

(0 <= L + 9) AND

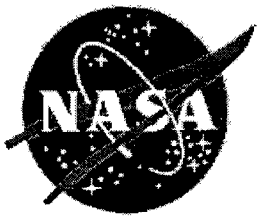
(L + 9 <= 10)

6 *FORALL levels L*

(0 <= L + fill(L) - usage) AND

(L + fill(L) - usage <= 10)

Analysis by Proof*

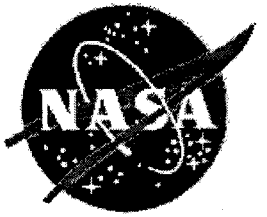


A Simple Formal Methods Example

- The following statements (statements 5.1 and 5.2) constitute an informal proof that the first FORALL statement (statement 5) is true:
- Property: *"5" FORALL levels L (L <= 1) IMPLIES THAT*
  (0 <= L + 9) AND (L + 9 <= 10)
- Proof:
  - *5.1 L+9 >= 0 because L >= 0 (and the sum of any two numbers greater than zero is greater than zero)*
  - *5.2 L+9 <= 10 because L <=1 (and any number less than or equal to 1 plus 9 is less than or equal to 10)*

**NOTE: Formal methods includes several other analysis techniques which don't require proofs. Additionally, many tools include automated provers which facilitate this type of analysis, if it is used.*

Why Can't I Get the Prover to Verify Property #6?

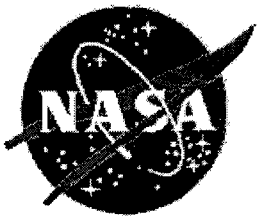


A Simple Formal Methods Example

- Property *"6"* *FORALL levels L*
  (0 <= L + fill(L) - usage) AND
  (L + fill(L) - usage <= 10)
- Proof Attempts Fails!
- Counter-example*
  - » Consider the case when L is 9:
    L + fill(L) - 1 = L+L-1 = 9+9-1 = 17 *(which is not <= 10)*
- The specification is flawed and must be corrected.

**NOTE: Some FM/AV tool (particularly model checkers) will automatically generate counter-examples*

Correcting the Specification



- Upon closer examination, it is found that statement 4, our expression for the water level at the next cycle, is in error:

  4 *level = L + fill(L) - usage* ***(incorrect)***

- This statement is inconsistent with the definition of fill because fill returns the new level of water, not just the amount of water added. The (corrected) expression for level, denoted by 4', is simply:

  4' *level = fill(L) - usage* ***(correct)***

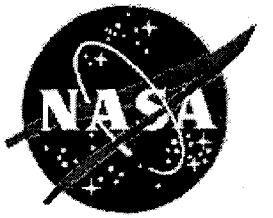
- The (corrected) FORALL statement (statement 6) is:

  6' *FORALL levels L:*

  (0 <= fill(L) - usage) AND

  (fill(L) - usage <= 10) ***(correct)***

This Simple Example Illustrates:

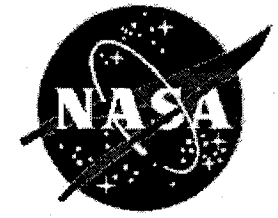


- **Formal Specification:** Modeling informal English statements using mathematical expressions
- **Type Checking:** Checking that all types of items are used consistently (e.g., ***level***)
- **Stating Properties:** Identifying and defining expected behavior of the system (e.g., the expected new level in the tank)
- **Proving Logical Conditions:** Constructing logical proofs which show that a given condition holds under all possible situations

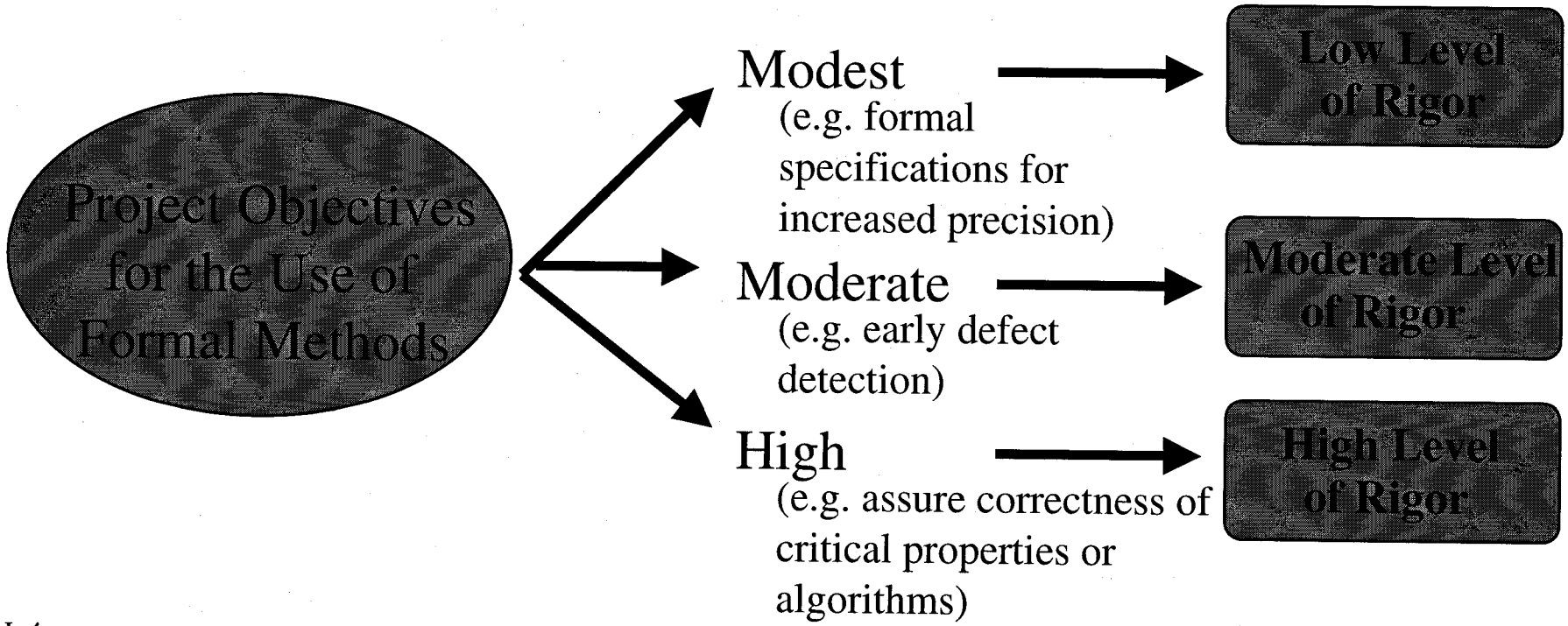
**NOTE: In the example, the name chosen for the "fill" function in statement 3 is misleading because the function returns the "actual level" rather than the "amount added". Statement 4, although wrong, is consistent with the casual reader's expectations, so the error is easy to overlook. For a more involved example consult the NASA SAFER system described in NASA-GB-001-97.*

[NASA-GB-002-95]

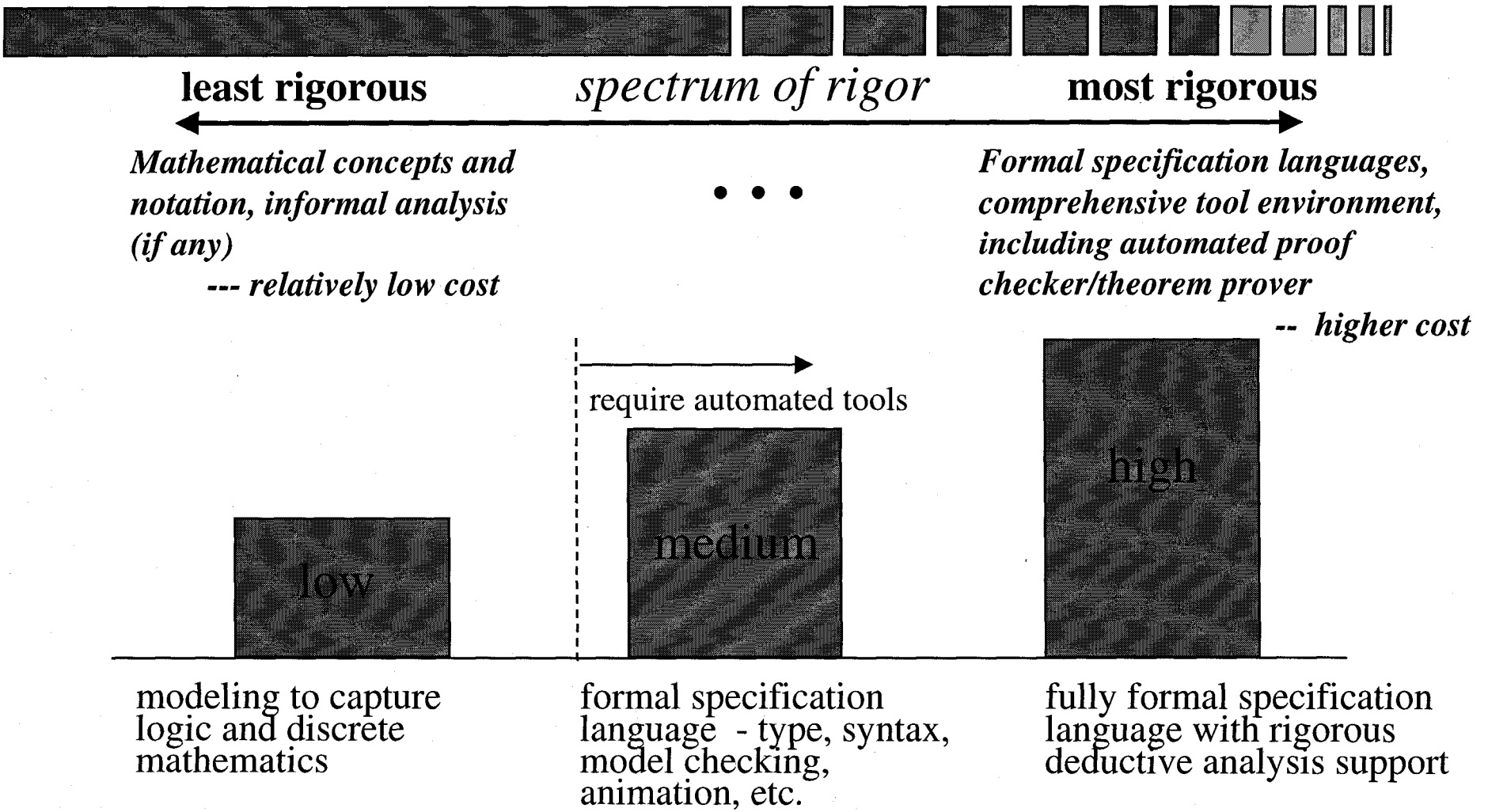
Types of Analysis/Formal Methods



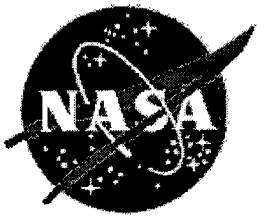
The preferred type of analysis and method is strongly influenced by the project objectives



Levels of Rigor in the Application of Formal Methods

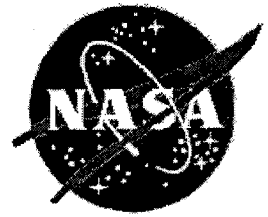


The "Method" Underlying Formal Methods



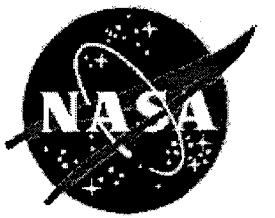
- **The Characterization Phase:** Synthesize a thorough understanding of the application and the application domain.
  - » Thorough study of the application
  - » Identify and study related work
  - » Acquire additional knowledge as needed
  - » Integrate the accumulated knowledge into a working characterization of the application

The "Method" Underlying Formal Methods (cont.)



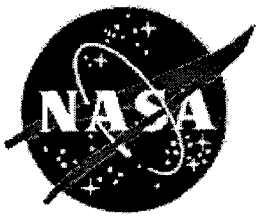
- **The Modeling Phase:** Define a mathematical representation suitable for formalizing the application domain and for calculating and predicting the behavior of the application in that context.
  - » Evaluate potential mathematical representations in the context of the underlying formal language and tools
  - » Select the most suitable mathematical representation
  - » Model key elements of the application and their relationships

The "Method" Underlying Formal Methods (cont.)



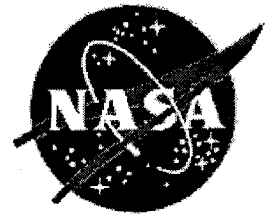
- **The Specification Phase:** Formalize relevant aspects of the application and its operational environment.
  - » Develop a specification strategy, considering such factors as hierarchical (multilevel) versus single-level specification, constructive versus descriptive specification style, procedural and organizational issues (such as developing reusable theories and common definitions), and specification chronology
  - » Using the chosen model and specification strategy, compose the specification
  - » Analyze the syntactic and semantic correctness of the specification

The "Method" Underlying Formal Methods (cont.)



- **The Analysis Phase:** Verify the specification. Including a subset of the following analysis techniques:
  - » Interpret or execute the specification
  - » Run high level scenarios on the animation
  - » Run model checker
  - » Prove key properties and invariants
  - » Establish the consistency of axioms, if any
  - » Establish the correctness of hierarchical layers, if any

The "Method" Underlying Formal Methods (Cont.)



- **The Documentation Phase:**
  - » Record operative assumptions, motivate critical decisions, document the rationale and crucial insights, provide explanatory material, trace specification to requirements (high-level design), track level of effort, and where relevant, collect cost/benefit data.

Process Considerations

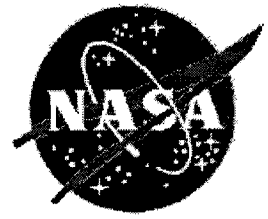
Development process should be "healthy," specifically, the process should have:

- » well defined phases
- » work products defined for each phase
- » analysis procedures in place for work products
- » scheduled review and v&v of work products

Process

- Healthy — ***good candidate for formal methods***
- Weak Process & Project is in trouble — ***not a good candidate, likely more benefit from process improvement measures***

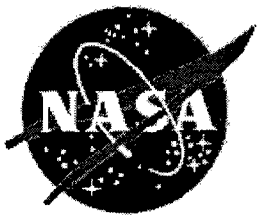
Placement of Formal Modeling and Analysis in a Development Lifecycle



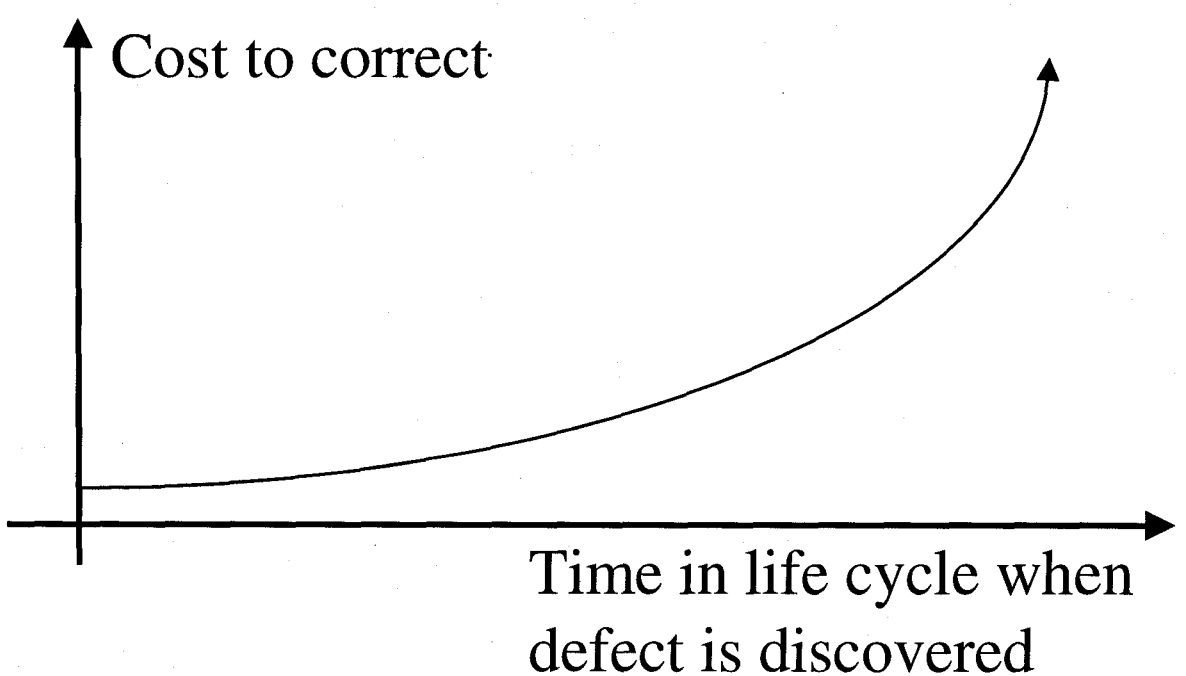
Why does this course focus on software requirements and design rather than code?

- The highest density of major defects found through the use of software inspections was during the requirements phase. This was several times higher than the density of major defects found in code inspections [Kelly92]
- Most hazardous software safety errors found during system integration and test of two NASA spacecraft were the result of requirements discrepancies or incorrect interface specifications [Lutz93]

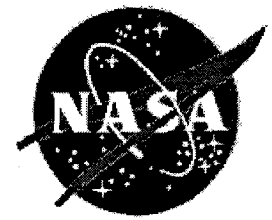
Defects and Cost to Correct



Earlier detection of defects is essential for complex projects

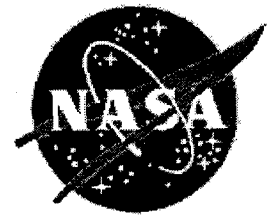


Limitations to Formal Methods



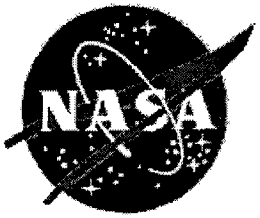
- Used as an adjunct to, not a replacement for, standard quality assurance methods
- Formal methods are not a panacea, but can increase confidence in a product's reliability if applied with care and skill
- Very useful for consistency checks, but can not assure completeness of a specification

Cautions in the Use of Formal Methods



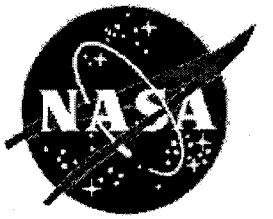
- Judicious application to suitable project environments is critical if benefits are to exceed costs
- Formal methods and problem domain expertise must be leveraged to achieve the best results

Formal Methods are Rapidly Being Adopted by Microprocessor & Computer Companies



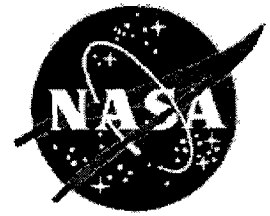
- Intel's Merced Project (Intel's next-generation microprocessor)
- National Semiconductor (VLSI design technologies to support advanced products)
- Hewlett-Packard (definition, design & verification of mid-range computer servers and high-end workstations)
- IBM (verification tools to support processors and systems)
- Texas Instruments (verification technologies and methodologies to support TI's design activities)

Summary



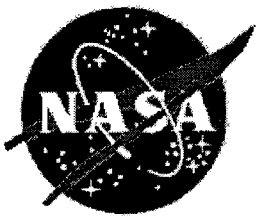
- Formal methods include a number of concepts/methods
  - » Abstraction
  - » Formal specifications
  - » Model checking
  - » Formal proofs
- Formal methods can be applied at various levels of rigor

Summary (cont.)

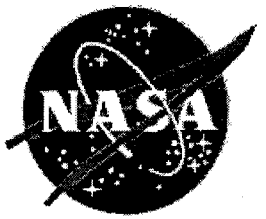


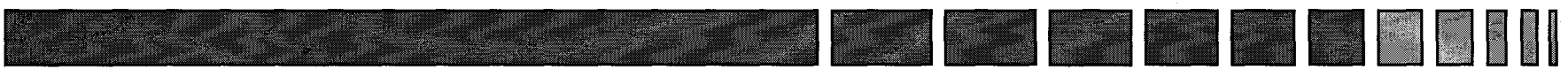
- Formal methods should be employed in a healthy development environment
- The five step "method" for performing formal analysis is: Characterization, Modeling, Specification, Analysis, and Documentation)
- Formal Methods is not a substitute for other QA activities
- Formal Methods be used judiciously in the proper domain

Select Resources



- SRI International Computer Science Laboratory
  - » http://www.csl.sri.com/sri-csl-fm.html
- Furman University - Formal Methods
  - » http://s9000.furman.edu/FM/
- Bell Labs - SPIN Model Checker
  - » http://netlib.bell-labs.com/netlib/spin/whatispin.html
- Formal Methods Virtual Library
  - » http://www.comlab.ox.ac.uk/archive/formal-methods.html
- Formal Methods Europe
  - » http://www.csr.ncl.ac.uk/projects/FME/index.html

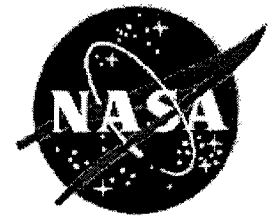




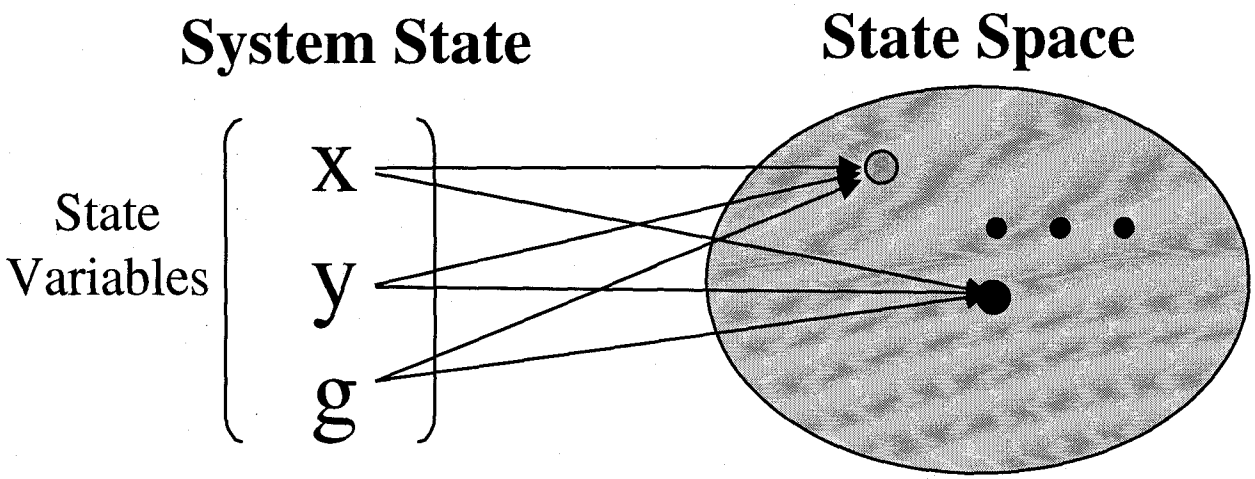
Backup (BU) Slides

Model Checking Details

BU1 - Finite State Models and State Space

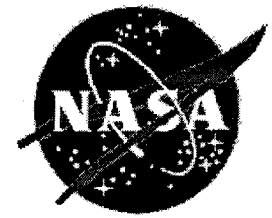


- System state can be modeled by a collection of state variables and their associated values
- State space represents the full range of values assumed by the state variables

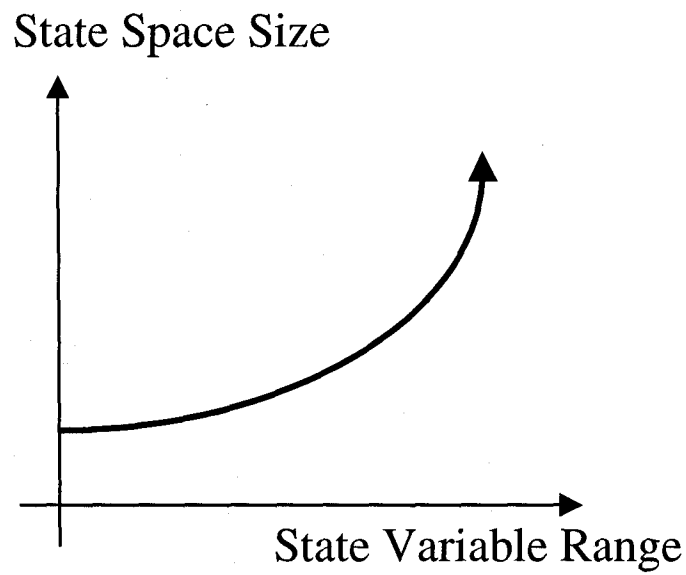


Each "point" in the state space represents one possible combination of state variable values.

BU2 - State Space Size

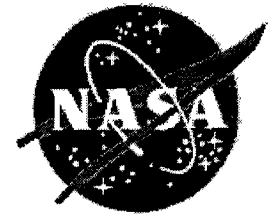


- The state space can be infinite, although for discrete systems we usually have a finite space state
- Even if finite, the state space is often intractably large
- State space size often grows rapidly (exponential or worse) with state variable range



State Space Explosion Problem

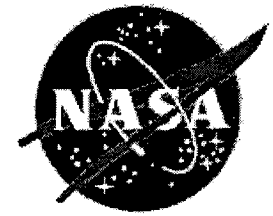
BU3 - State Space Size -- Example



In an example where we are assigning instruments to I/O channels, the state space consists of all possible relations from the set of instruments to the set of channels. The number of these relations is $2^{n \times m}$, where n = number of instruments and m = number of channels.

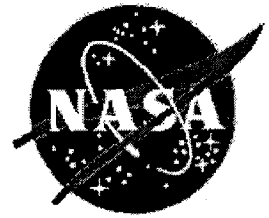
Even for a modest-sized problem, say 8 instruments and 5 channels, the state space contains 2^{40} possible elements. [In this case not all of these states may be reachable since both the number of instruments per channel and the number of channels per instrument may be limited.]

BU4 - Model Checkers

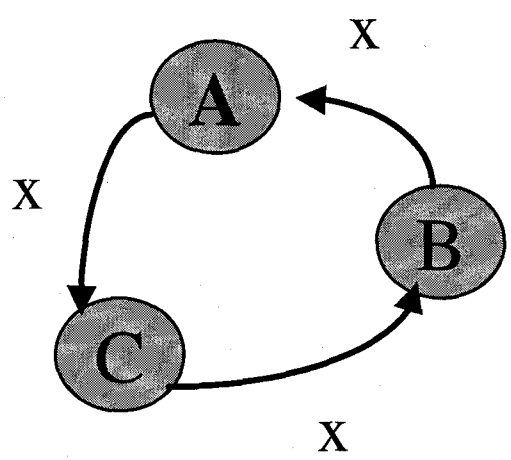


- Verification systems that perform logic model checking are referred to as model checkers. Objective is to verify a model over its corresponding state space (the subset of reachable states).
- Basic function of a model checker is to determine whether a given finite state model of a system's requirements satisfies a formula in a given logic.
- Model checkers are operational as opposed to analytic.
- Models are expressed in a suitable language (e.g. SMV, Murphϕ, PROMELA(SPIN)).
- Can be used on suitably restricted "partial specifications".
- Usually, the goal is to find errors as opposed to proving correctness.

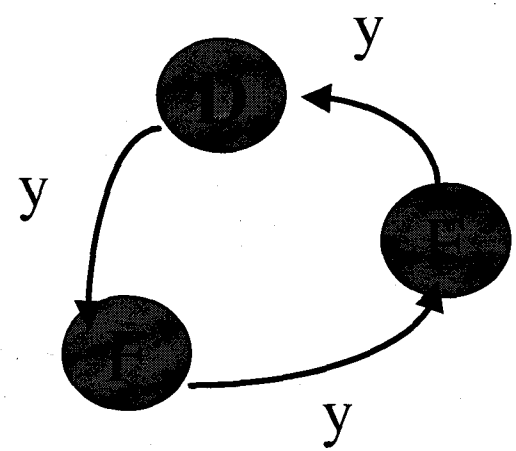
BU5 - Model Checking and Computational Trees



Consider two concurrent processes P1 and P2 depicted by the following state machine diagrams (example adapted from Callahan*)



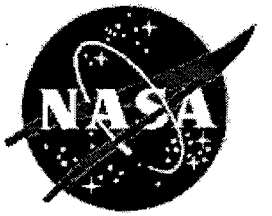
Process P1



Process P2

*J. Callahan, *Automated Testing via Model Checking*, presentation.

BU6 - Model Checking and Computational Trees (cont'd)



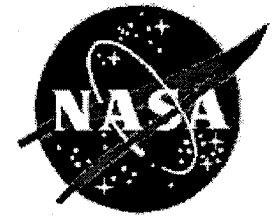
Considered together, the joint process machine has nine states (the Cartesian product of the state space for P1 with the state space for P2):

(A,D)	(B,D)	(C,D)
(A,E)	(B,E)	(C,E)
(A,F)	(B,F)	(C,F)

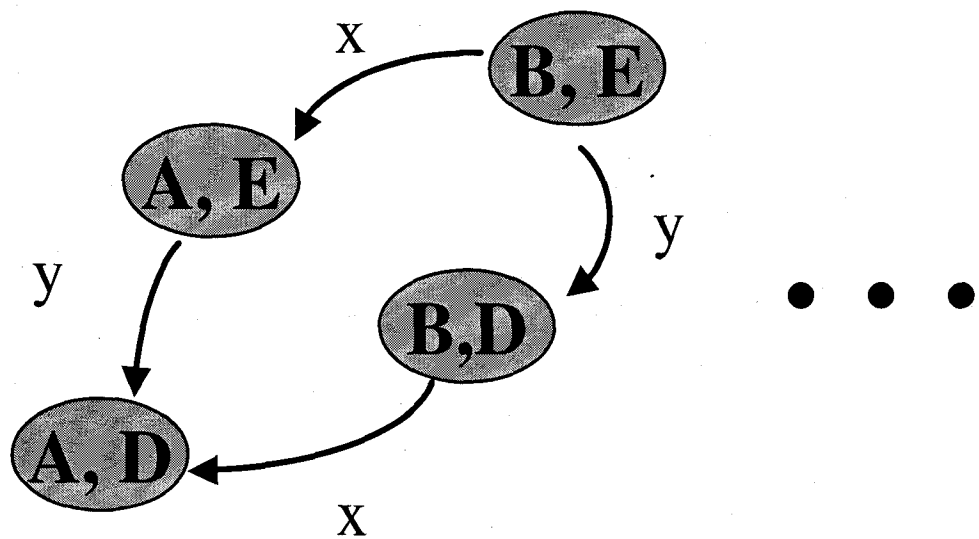
State Space for Joint Process Machine

L 5 J. Callahan, *Automated Testing via Model Checking*, presentation.

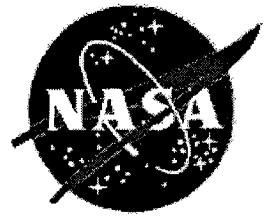
BU7 - Model Checking and Computational Trees (cont'd)



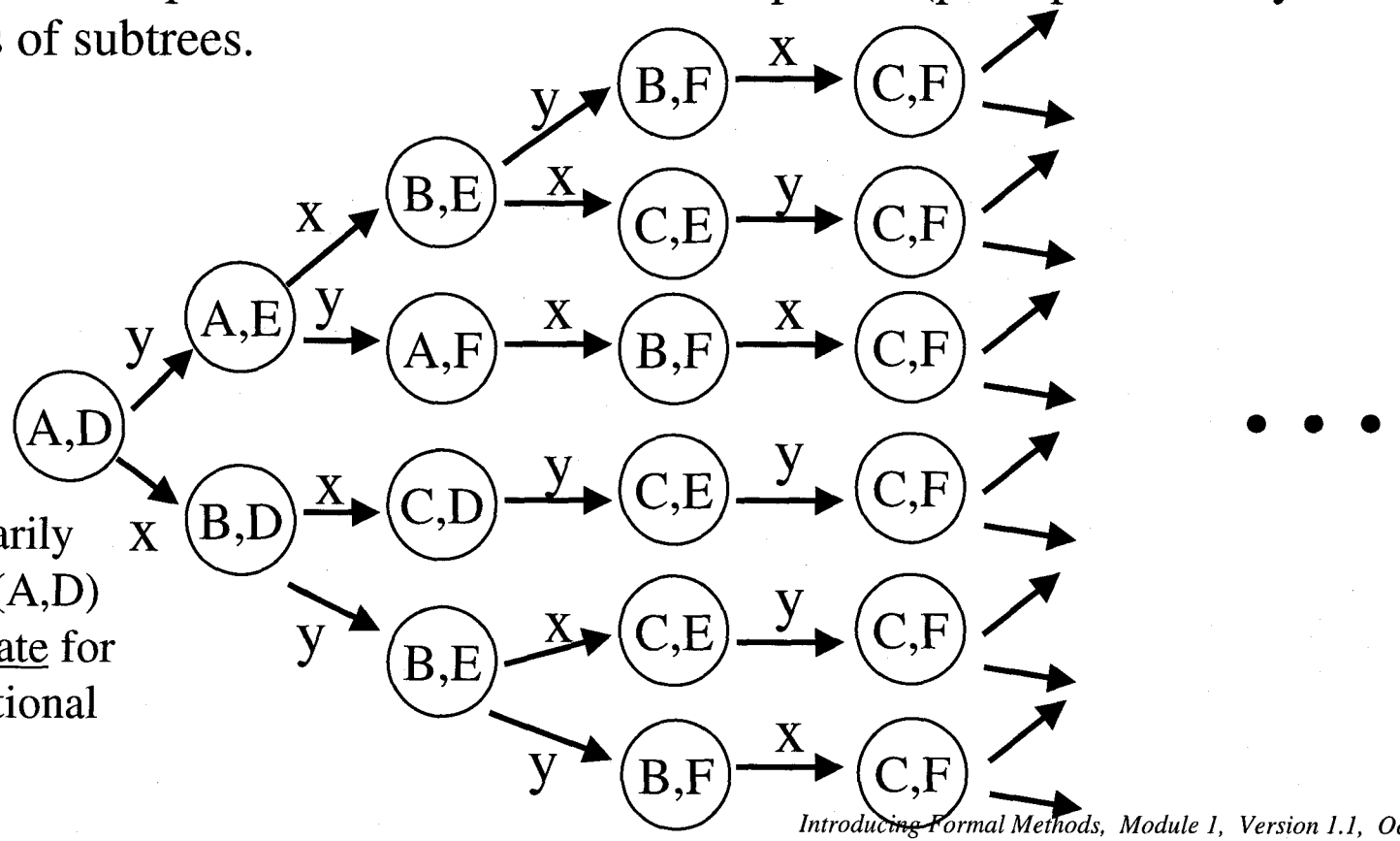
A **partial** state transition diagram for the joint process machine.



BU8 - Model Checking and Computational Trees (cont'd)



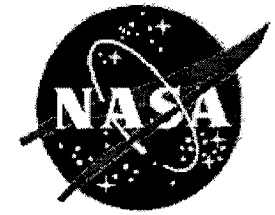
Model checkers effectively and automatically explore ALL paths from a start state in a computational tree. Note we have not listed in the tree any state that was reached in a prior level. The computational tree will contain repeated (perhaps infinitely many times) copies of subtrees.



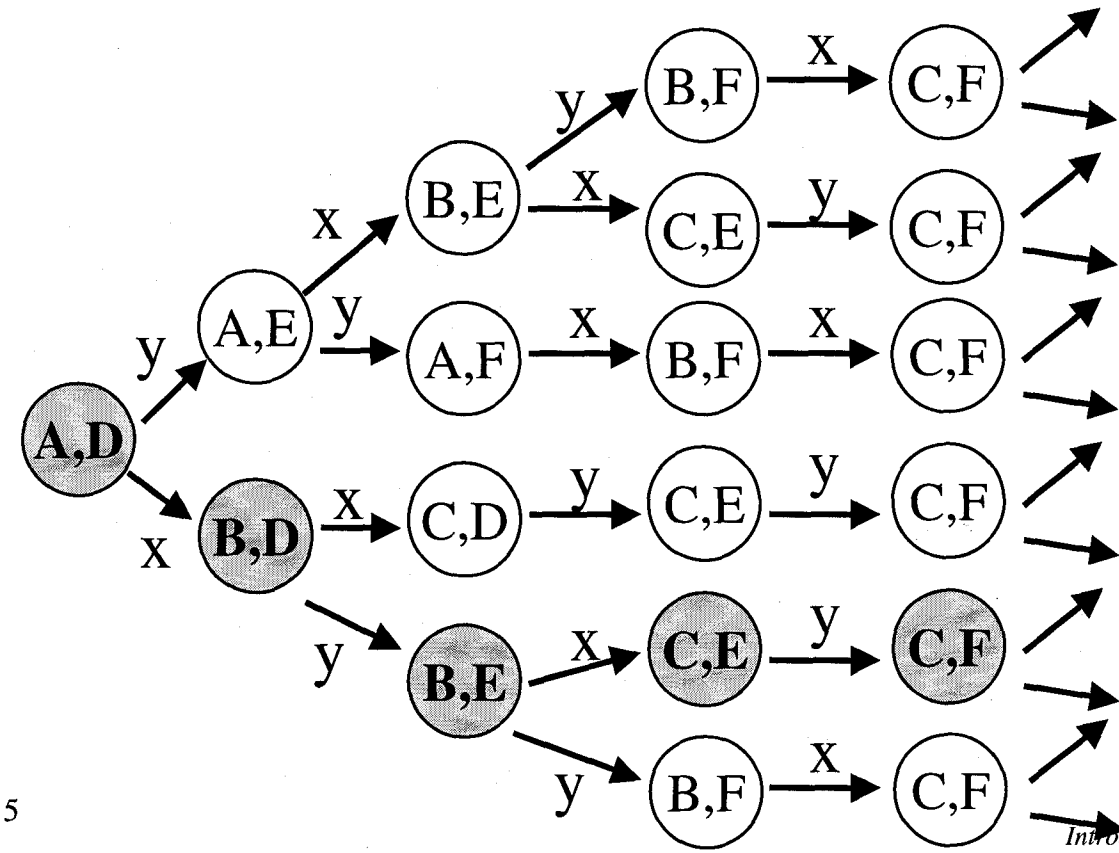
We've arbitrarily chosen state (A,D) as the start state for this computational tree.

L 5

BU9 - Model Checking and Computational Trees (cont'd)

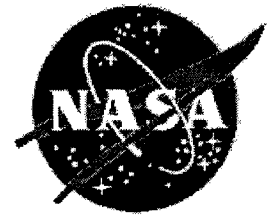


A string of symbols $\sigma = s_1 s_2 \ldots$ where each s_i is chosen from the union of the two original state machine input alphabets, is called an input trace. Model checkers will employ input traces that exercise all reachable states.



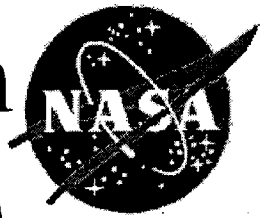
Using start state AD, trace σ = xyxy reaches states BD, BE, CE, and CF in sequence.

BU10 - Approaches for Coping with the State Explosion Problem



- Use a symbolic, rather than explicit, representation of the state space
  - » a set of states is represented by a logical formula that is satisfied in a given state if and only if the state is a member of the set
  - » most widely used representation is known as Ordered Binary Decision Diagrams (or OBDD's -- sometimes just BDD's)
- Use symmetries of the state space to reduce the number of states to be considered

BU11 - Approaches for Coping with the State Explosion Problem (cont'd)



- Use methods to reduce a verification problem to a series of manageable sub-problems
- Use partial order reduction methods reducing the size of the checked state space
- Reduce verification problem to a *partial specification*, eliminating irrelevant states